

FernUniversität Hagen
Fachbereich Informatik
Lehrgebiet Praktische Informatik IV

Wintersemester 2001

Software-Praktikum 1585
„Mobiler digitaler Stadtführer“

September 2001

Inhaltsverzeichnis

1	Einleitung und Motivation der Aufgabenstellung.....	1
2	Der Wegenetzeditor.....	2
2.1	Überblick	2
2.2	Das Wegenetz	2
2.3	Interessante Punkte im Wegenetz	4
2.4	Dateioperationen und ein Beispiel.....	4
3	Die Wegenetzdatei	5
3.1	Überblick	5
3.2	XML.....	6
3.3	Die Wegenetz-DTD	7
3.4	Eine Wegenetzdatei als Beispiel.....	9
4	Der mobile digitale Stadtführer.....	11
5	Anfrageoperationen.....	12
5.1	Relevante Daten für die Anfragekomponente	12
5.2	Datendefinition	13
5.3	Anfragearten	14
5.4	Statische Anfragen.....	14
5.5	Permanente Anfragen	16
5.6	Getriggerte Anfragen	17
5.7	Zurücknehmen von Anfragen	18
5.8	Nacheinanderausführung von Anfragen	18
5.9	Das FINDE-Kommando	19
5.10	Das FÜHRE-Kommando	19
5.11	Auswahl von Teilnetzen	19
5.12	Scrollmechanismus für Textfenster	20
5.13	Ansichtsarten sowie Vergrößerungs- und Verkleinerungsoperationen	20
6	Anforderungsdefinitions- und Entwurfsrichtlinien	20
7	Programmierrichtlinien und Dokumentation	21

1 Einleitung und Motivation der Aufgabenstellung

Die allgemeinen Ziele des Software-Praktikums („Lernziele“) bestehen unter anderem darin, anhand einer komplexeren Aufgabenstellung

- gelernte Methoden der Softwareentwicklung in eingeschränktem Maße umzusetzen,
- ein „mittelgroßes“ Softwareprojekt zu organisieren und durchzuführen,
- Teamarbeit, Konzeption, Planung und Realisierung eines Projekts im Rahmen einer Gruppe kennenzulernen,
- auftretende Probleme zu analysieren und zu zerlegen,
- entsprechende Lösungsansätze zu finden,
- „strukturierte Programmierung im Größeren“ zu betreiben,
- eine benutzerfreundliche Schnittstelle zu entwickeln und
- eine persistente Datenhaltung zu realisieren.

Im Gegensatz zum Programmierpraktikum aus dem Grundstudium werden wir im Software-Praktikum nicht bis ins Detail vorgeben, auf welche Art und Weise Teilaufgaben zu lösen sind, sondern es wird eine Art von „Produktfunktionalität“ spezifiziert, welche das zu erstellende Softwaresystem erfüllen muß. Dies gibt den einzelnen Gruppen Freiheitsgrade zur Lösung und konkreten Realisierung von Teilproblemen und Spielraum für eigene Kreativität und Ideen.

Thematisch befaßt sich das Praktikum mit einer Simulation eines mobilen, digitalen Stadtführers (MDS). Dabei ist an ein Programm für Palmtops gedacht, das die Funktionen eines Stadtführers und eines Routenplaners vereint, um die Navigation in einer unbekanntem Stadt zu erleichtern. Wir unterstellen, daß jede Stadt den eigenen Stadtplan (mit Straßennamen und mit vielen Zusatzinformationen über besonders interessante Punkte in der Stadt, wie z.B. Museen, Kirchen, Restaurants) in unserem Format zur Verfügung stellt. Der Benutzer des Programms kann sich nun vor dem Besuch der Stadt diese Informationen auf seinen Palmtop laden und so seinen Aufenthalt planen und sich vor Ort informieren und führen lassen.

Der simulierte Palmtop ist mit einem GPS (Global Positioning System) ausgestattet, so daß die aktuelle Position des Benutzers bekannt ist. Auf dem Palmtop wird nun standardmäßig eine (zoombare) Karte der Stadt angezeigt, in der man sich gerade befindet. Die eigene Position (ermittelt durch GPS) wird auf dieser Karte markiert. Nun erlaubt unser Stadtführer, Anfragen zu stellen. Diese Anfragen sollen gesprochene Worte sein, die dann ausgewertet werden. Die Antwort wird dann, je nach Anfrage, ebenfalls gesprochen ausgegeben oder auf dem Palmtop angezeigt. In unserer Simulation wird das gesprochene Wort allerdings durch textuelle Ein- bzw. Ausgabe ersetzt.

Alle akzeptierten Anfragen beziehen sich auf die zur Stadt gespeicherten Informationen zum Wegenetz mit den dortigen, besonders interessanten Punkten. Zusätzlich ist es möglich, Anfragen zu anderen Nutzern des Programms zu stellen, die dann als dynamische Punkte angezeigt werden können. Durch die verschiedenen Anfragekommandos können z.B. Informationen über nahegelegene Geschäfte, Aufenthaltsorte von Personen oder auch Wege zu speziellen Zielen ermittelt werden. Die Anfragesprache erlaubt zudem die Definition von Objektgruppen, die für den Benutzer besonders interessant sind.

Die Aufgabe im Software-Praktikum besteht aus zwei Teilaufgaben. Die erste Aufgabe umfaßt die Erstellung eines Wegenetzeditors (siehe Abschnitt 2), mit dem sowohl das Wegenetz einer Stadt als auch die Klassen für die besonders interessanten Punkte eingegeben werden können. Die zweite Teilaufgabe beinhaltet die simulierte Palmtop-Anwendung. Zentrale Teile sind hierbei die Anzeige der Karte mit den relevanten Daten (Abschnitt 4) und die Anfragesprache (Abschnitt 5.1 und Abschnitt 5).

2 Der Wegenetzeditor

2.1 Überblick

Im ersten Teil des Projektes wird der Wegenetzeditor erstellt. Er wird dazu verwendet, die Daten für den Stadtführer zu generieren. Dazu verfügt er über eine graphische Benutzerschnittstelle, die es dem Benutzer erlaubt, das Wegenetz auf einfache Weise einzugeben und zusätzlich besonders interessante Punkte mit ihren Attributen und einer Beschreibung zu definieren.

Die Arbeit mit diesem Programm kann man sich z.B. so vorstellen, daß zunächst eine Hintergrundkarte z.B. als JPG-Datei geladen und dargestellt wird. Die abgebildeten Wege (dies können u.a. Fußwege, Straßen oder Buslinien sein) werden dann vom Anwender nachgezeichnet. Anschließend gibt es dann die Möglichkeit, Punkte einzugeben, die die unterschiedlichen Geschäfte, Sehenswürdigkeiten und sonstigen interessanten Orte repräsentieren.

Die erzeugten Daten werden schließlich in eine Datei im XML-Format gespeichert.

2.2 Das Wegenetz

Das Wegenetz wird durch einen Graphen repräsentiert, der eine Sorte von Knoten, aber verschiedene, vordefinierte Sorten von Kanten hat. Diese sind:

- Fußweg
- (Auto-) Straße
- Autobahn
- Buslinie
- Bahnlinie

Die Eingabe erfolgt mit der Maus, indem die Wege (Straßen, Buslinien usw.) nachgezeichnet werden. Alle Wege bestehen aus einer Menge von aneinanderhängenden Liniensegmenten, einem Startknoten, einem Endknoten und Stützpunkten dazwischen.



Es können sowohl gerichtete als auch ungerichtete Kanten gezeichnet werden. Eine ungerichtete Kante wird intern durch zwei gerichtete Kanten dargestellt. Bei der Eingabe dient dieser Mechanismus der Übersichtlichkeit und Einfachheit.

Der Benutzer ist gezwungen, einen der vordefinierten Typen zu wählen, um eine Kante zu zeichnen. So wird sichergestellt, daß jede Kante einen Typ hat. Zu jeder Kante können weitere Informationen angegeben werden, nämlich ein Name (dieser wird z.B. für den Straßennamen oder die Nummer einer Buslinie verwendet) und weitere, beschreibende Informationen.

Knoten des Graphen haben keinen Typ. In ihnen dürfen Kanten beliebigen Typs beginnen und enden. Zu Knoten können ebenfalls ein Name und weitere Informationen angegeben werden.

Bei der Eingabe eines Wegenetzes müssen einige Regeln beachtet werden:

- Eine Kante verbindet stets zwei Knoten. Stützpunkte sind (zunächst) keine Knoten.
- Beim Zeichnen einer Kante werden Start- und Endknoten als neue Knoten erzeugt, sofern sie nicht bereits auf existierenden Knoten liegen.
- Ein Stützpunkt einer Kante a wird zum Start- bzw. Endknoten einer Kante b , wenn b in diesem Stützpunkt beginnt bzw. endet. Die Kante a wird damit in zwei neue Kanten a_1 und a_2 unterteilt.
- Wenn sich zwei Kanten kreuzen, wird am Kreuzungspunkt nicht automatisch ein neuer Knoten erzeugt. Wenn dort ein Knoten gewünscht wird, muß dieser manuell erzeugt werden.
- Ein Knoten, der auf einem Liniensegment oder einem Stützpunkt einer Kante a erzeugt wird, teilt diese in a_1 und a_2 . Die Kanten a_1 und a_2 besitzen dieselben Eigenschaften (Name, Typ und Informationen) wie a .
- Eine Kante kann nur einen Typ haben. Wenn zwei Kanten verschiedenen Typs denselben räumlichen Verlauf haben, so sind beide Kanten einzuzuzeichnen.

Bei der Eingabe des Wegenetzes wird von einem Snap-Mechanismus Gebrauch gemacht, der das Übereinanderlegen von Knoten und das Aufeinanderlegen von Knoten und Kanten vereinfacht. Dabei wird der Mauszeiger automatisch auf einen Knoten oder eine Kante „gezogen“, wenn er in die Nähe dieses Knotens oder dieser Kante kommt. Dieser Effekt kann über einen Menüpunkt an- und ausgeschaltet werden.

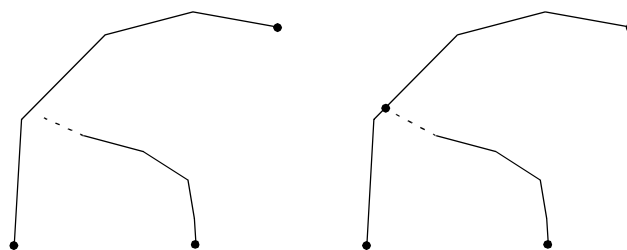


Abbildung 1: Wird bei der Eingabe einer neuen Kante die Maus in die Nähe einer bereits existierenden Kante bewegt, so tritt der Snap-Mechanismus in Kraft: Er bietet an, einen neuen Knoten auf der Kante zu erzeugen. In gleicher Weise gilt dies bei der Erzeugung neuer Knoten.

Wenn mehrere, räumlich übereinanderliegende Kanten für den Snap zur Auswahl stehen, dann wird zunächst ein Snap mit einer Kante desselben Typs angeboten. Desweiteren gibt es eine Möglichkeit, auch beliebige Kanten zum Snap auszuwählen.

Die Koordinaten aller Punkte werden durch das Hintergrundbild (die Karte) festgelegt. Der Punkt (0, 0) liegt stets unten links, während der Punkt oben rechts durch die Auflösung des Bildes bestimmt wird. Besitzt das Bild 300*200 Bildpunkte, so erhält der Punkt oben rechts die Koordinaten (299, 199).

2.3 Interessante Punkte im Wegenetz

Zusätzlich zur Eingabe des Wegenetzes können noch weitere Punkte angegeben werden. Dies sind besonders interessante Punkte wie z.B. Restaurants, Museen oder historische Bauwerke. Um solche Punkte eingeben zu können, muß zunächst eine Klasse für jede spezielle Art der Punkte definiert werden. Eine Klasse kann dann beliebig viele Attribute haben. Attribute sind vom Typ *int*, *real*, *string*, *bool* oder sie sind Aufzählungstypen (*enum*). Die Definition einer solchen Klasse könnte in einer Programmiersprache beispielsweise wie folgt aussehen:

```
Klasse Kirche {
    Name : string;
    Baustil : string;
    Baujahr : int;
    Architekt : String;
    Religion : katholisch | evangelisch;
    Bischofssitz: bool;
}
```

Im Editor werden Klassen allerdings mit Hilfe eines komfortables Dialogfensters eingegeben und in einer internen Datenstruktur gespeichert.

Nachdem Klassen für interessante Punkte definiert wurden, können solche Punkte im Wegenetz angelegt werden. Dazu ist zunächst eine Klasse auszuwählen. Dann wird mit der Maus die Position gewählt, an der der Punkt eingetragen werden soll. Da ein solcher Punkt *immer* einen Zugang zum Wegenetz braucht, muß außerdem ein Anschluß zu einem Knoten oder einer Kante hergestellt werden. Ein Punkt kann mehrere Anschlüsse haben. Desweiteren ist der Benutzer gezwungen, alle Attribute, die die Klasse des Punktes hat, anzugeben.

Auf die beschriebene Art und Weise werden das Wegenetz und alle interessanten Punkte angegeben. Es werden allerdings keinerlei Informationen über die Darstellung der einzelnen Komponenten gemacht, da für die Darstellung ausschließlich die Simulation des Palmtops zuständig ist. Um eine gewisse Übersichtlichkeit bei der Eingabe zu gewährleisten, sind den unterschiedlichen Kantentypen trotzdem verschiedene Farben zugeordnet.

2.4 Dateioperationen und ein Beispiel

Schließlich gibt es noch einige I/O-Operationen: Neben dem bereits erwähnten Laden einer Karte werden das Importieren und Exportieren von Wegenetzen angeboten. Die Daten zum Wegenetz

werden in einem XML-Format gespeichert (Abschnitt 3) während das Bild der Karte unverändert abgelegt wird.

Abschließend hier noch ein Beispiel für ein Wegenetz, wie es mit dem Editor erzeugt werden kann.

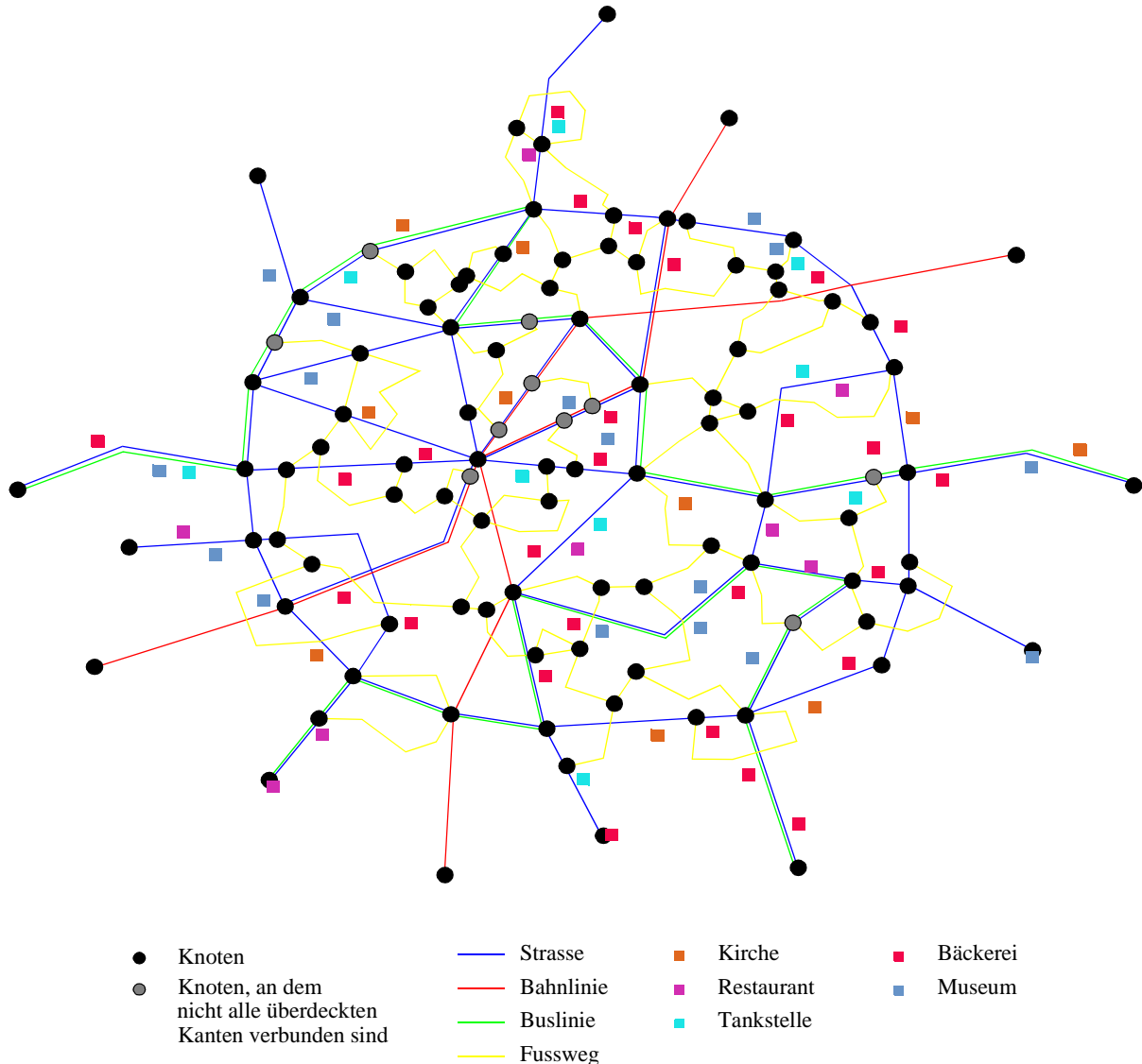


Abbildung 2: Die Abbildung zeigt ein Wegenetz mit verschiedenen Wegtypen und einigen interessanten Punkten. Nicht gezeigt werden hier die Anschlußlinien dieser Punkte zu den Kanten und Knoten.

3 Die Wegenetzdatei

3.1 Überblick

Die Wegenetzdatei dient als Schnittstelle zwischen dem Wegenetzeditor aus Abschnitt 2 und dem mobilen digitalen Stadtführer, der in Abschnitt 4 genauer beschrieben wird. Ihr Format ist fest vorgegeben, um die Austauschbarkeit der Programme und Daten unter den verschiedenen Sopro-Gruppen zu gewährleisten und das Erzeugen solcher Dateien auch Externen zu ermöglichen.

Die Wegenetzdatei soll im XML-Format erstellt werden. XML bietet eine Fülle von Möglichkeiten, Daten strukturiert in einer Textdatei zu beschreiben. Man muß jedoch kein XML-Experte sein, um diese Sopra-Aufgabe zu absolvieren. Eine (sehr kurze) Einführung in die Thematik gibt Abschnitt 3.2; darüber hinaus existiert mittlerweile umfangreiche Literatur zum Thema. Als ein Beispiel sei hier das Buch „XML in der Praxis“ von Behme und Mintert genannt, das auch online verfügbar ist (<http://www.mintert.com/xml/buch/>). Zur weitergehenden Einarbeitung in das Thema sind die XML-Seiten des World-Wide-Web-Consortiums (W3C) ein empfehlenswerter Ausgangspunkt (<http://www.w3.org/XML>).

Die Verwendung von XML als Basisformat bietet mehrere Vorteile gegenüber einem ganz eigenen (Binär-)Format. Da XML ein ASCII-Format ist, sind die Dateien mit beliebigen Editoren lesbar und veränderbar, zudem existiert mittlerweile eine Vielzahl von speziellen XML-Editoren. Für die Implementierung bietet XML außerdem den großen Vorteil, daß kein eigener Parser geschrieben werden muß, da leistungsfähige Java-Bibliotheken bereits im Java Development Kit enthalten sind. Weitere Informationen hierzu finden Sie unter http://java.sun.com/xml/tutorial_intro.html.

3.2 XML

Eine XML-Datei beschreibt ein *Element*. Dieses Element beginnt mit einem *Start-Tag* und endet mit einem *Ende-Tag* (engl. „tag“ = „Marke“). Ein Start-Tag beginnt mit einer „<“-Klammer, nennt dann einen Tag-Namen, und endet mit einer „>“-Klammer. Das dazugehörige Ende-Tag sieht genauso aus; allerdings wird dem Tag-Namen ein Schrägstrich („/“) vorangestellt. Den Tag-Namen kann man auch als *Typ* des Elementes interpretieren.

Zwischen Start- und Ende-Tag wird der *Inhalt* des Elementes beschrieben. Dies kann zum einen eine Zeichenkette, die wir auch als *Wert* bezeichnen, und zum anderen eine beliebige Anzahl von weiteren Elementen sein. Darüber hinaus kann ein Tag mit zusätzlichen Informationen versehen werden, die *Attribute* genannt werden. Nachfolgend ist ein Beispiel für eine XML-Datei aufgeführt (die erste Zeile ist der Standardkopf für XML-Dateien):

```
<?xml version="1.0"?>
<Datum Feiertag = "Heiligabend">
  <Tag>
    24
  </Tag>
  <Monat>
    12
  </Monat>
  <Jahr>
    2001
  </Jahr>
</Datum>
```

Die verwendeten Tags heißen *Datum*, *Tag*, *Monat* und *Jahr*. Das Datum-Tag besitzt ein Attribut namens *Feiertag*, dessen Wert *Heiligabend* ist.

Um nun für konkrete Anwendungen genauer festzulegen, wie XML-Dateien strukturiert werden müssen, um als Eingaben verarbeitet werden zu können, gibt es den Mechanismus der *Document Type Description (DTD)*. Eine DTD gibt an, welche Tags existieren, welchen Inhalt diese Tags

haben dürfen und welche Attribute einem Tag zugeordnet werden dürfen oder müssen. Die DTD zu unserem Datumsbeispiel lautet:

```
<!ELEMENT Datum (Tag, Monat, Jahr)>
<!ATTLIST Datum Feiertag #IMPLIED>
<!ELEMENT Tag (#PCDATA)>
<!ELEMENT Monat (#PCDATA)>
<!ELEMENT Jahr(#PCDATA)>
```

Der Erläuterung bedürfen hier sicher die Schlüsselwörter *#PCDATA* und *#IMPLIED*. *#PCDATA* gibt an, daß in der XML-Datei an der entsprechenden Stelle beliebiger Text stehen darf, *#IMPLIED* besagt, daß das *Feiertag*-Attribut optional ist (im Gegensatz zu *#REQUIRED*). In der Wegenetz-DTD aus Abschnitt 3.3 verwenden wir noch einige weitere Konstrukte, deren Bedeutung Sie bitte der Literatur entnehmen.

3.3 Die Wegenetz-DTD

Dieser Abschnitt beschreibt die DTD, die das Format der im Software-Praktikum verwendeten Wegenetzdateien spezifiziert. Eine vollständige Wegenetz-Datenbank enthält folgende Informationen:

1. Name der Datei, die die Hintergrundkarte enthält. Hier können auch relative oder absolute Pfade angegeben werden.
2. Eine Menge von *Knoten*.
3. Eine Menge von *Kanten*, die jeweils zwei Knoten verbinden. Kanten repräsentieren Wege.
4. Eine Menge von *Klassen*. Eine Klasse spezifiziert die Attribute der Punkte aus (5.). Auf diese Attribute kann dann in Anfragen Bezug genommen werden.
5. Eine Menge von (interessanten) *Punkten*. Jeder Punkt gehört zu genau einer der Klassen aus (4.).

In der DTD ist ein Wegenetz wie folgt definiert:

```
<!ELEMENT WEGENETZ (KARTE, KNOTEN, KANTEN, KLASSEN, PUNKTE)>
```

Zur Spezifikation der Karte wird der Dateiname angegeben.

```
<!ELEMENT KARTE (#PCDATA)>
```

Die Mengen *KNOTEN*, *KANTEN*, *KLASSEN* und *PUNKTE* bestehen aus beliebig vielen Elementen des jeweiligen Typs.

```
<!ELEMENT KNOTEN (Knoten*)>
<!ELEMENT KANTEN (Kante*)>
<!ELEMENT KLASSEN (Klasse*)>
<!ELEMENT PUNKTE (Punkt*)>
```

Ein *Knoten* wird durch seine Koordinaten, also seine Position auf der Karte, spezifiziert. Außerdem muß ihm eine eindeutige *KnotenId* zugewiesen werden. Optional können einem Knoten ein Name und ausführlichere Informationen zugewiesen werden.

```
<!ELEMENT Knoten (Koordinaten, Name?, Info?)>
<!ATTLIST Knoten KnotenId ID #REQUIRED>
```

Zu einer *Kante* muß ihr Start- und ihr Zielknoten angegeben werden. Außerdem muß ihr eine eindeutige *KantenId* sowie ein *Typ* zugewiesen werden. Zulässige Typen sind:

- *Fussweg*: nur zu Fuß zu benutzen,
- *Strasse* und *Autobahn*: nur mit dem Auto zu benutzen,
- *Bahnlinie* und *Buslinie*: als Fußgänger zu benutzen, jedoch schneller.

Da in unserer Anwendung Knoten und Kanten räumlich eingebettet sind, repräsentiert eine Kante im allgemeinen einen Linienzug, nicht eine Strecke. Deshalb dürfen beliebig viele *Stützpunkte* zwischen Start und Ziel angegeben werden. Optional können einer Kante ein Name und ausführlichere Informationen zugewiesen werden.

```
<!ELEMENT Kante (Start, Ziel, Stuetzpunkt*, Name?, Info?)>
<!ATTLIST Kante
  KantenId ID #REQUIRED
  Typ (Fussweg|Strasse|Autobahn|Buslinie|Bahnlinie) #REQUIRED>
<!ELEMENT Start EMPTY>
<!ATTLIST Start KnotenId IDREF #REQUIRED>
<!ELEMENT Ziel EMPTY>
<!ATTLIST Ziel KnotenId IDREF #REQUIRED>
<!ELEMENT Stuetzpunkt (X, Y)>
```

Eine *Klasse* besteht aus einem Namen und einer beliebig großen Menge von *Attributen*. Außerdem muß ihr eine eindeutige *KlassenId* zugewiesen werden. Ein (Klassen-)Attribut ist von einem der Typen *int*, *real*, *bool*, *string* oder *enum*. Das Klassenattribut wird als (XML-)Attribut angegeben. Im Falle des Attributtyps *enum* handelt es sich um einen Aufzählungstyp, so daß zusätzlich Aufzählungskonstanten angegeben werden müssen.

```
<!ELEMENT Klasse (Name, Attribut*)>
<!ATTLIST Klasse KlassenId ID #REQUIRED>
<!ELEMENT Attribut (Name, Konstante*)>
<!ATTLIST Attribut Typ (int|real|bool|string|enum) #REQUIRED>
<!ELEMENT Konstante (#PCDATA)>
```

Ein (interessanter) *Punkt* wird durch seine Koordinaten, also seine Position auf der Karte, spezifiziert. Außerdem muß seine Klasse angegeben werden. Weiterhin wird bestimmt, wie er an das Wegenetz angeschlossen ist, welche Attributwerte er besitzt und wie sein Name lautet. Optional können einem Knoten ausführlichere Informationen zugewiesen werden.

Ein Punkt muß einen oder mehrere *Anschlüsse* an das Wegenetz haben. Ein Anschluß erfolgt entweder durch Zuordnung zu einer Kante oder zu einem Knoten des Netzes. Im ersten Fall muß zur Bestimmung der Position auf der Kante zusätzlich ein Wert zwischen 0 und 1 angegeben werden (0: direkt am Startknoten, 1: direkt am Zielknoten, >0 und <1: linear interpoliert).

```
<!ELEMENT Punkt (Koordinaten, Anschluss+, Attributwert*, Name, Info?)>
<!ATTLIST Punkt KlassenId IDREF #REQUIRED>
<!ELEMENT Anschluss ((KantenRef, KantenPos) | KnotenRef)>
<!ELEMENT KantenRef EMPTY>
<!ATTLIST KantenRef KantenId IDREF #REQUIRED>
<!ELEMENT KantenPos (#PCDATA)> <!-- Echter Typ: real zwischen 0 und 1-->
<!ELEMENT KnotenRef EMPTY>
<!ATTLIST KnotenRef KnotenId IDREF #REQUIRED>
<!ELEMENT Attributwert (#PCDATA)> <!-- Echter Typ: entsprechend der Klassenpezifikation -->
```

In dem obigen DTD-Fragment erscheint der Kommentar `<!-- Echter Typ: real -->`. Dies ist nötig, da man in DTDs leider nur den Datentyp `#PCDATA`, also *String*, verwenden kann. Oft ist es jedoch wünschenswert, den Typ eines Elementes genauer zu bestimmen. In solchen Fällen fügen wir einen entsprechenden Kommentar ein. Solche Zusatzspezifikationen sind genauso bindend für die Korrektheit der entsprechenden XML-Dokumente wie die regulären DTD-Spezifikationen.

Schließlich folgen noch einige Tags, die an mehr als einer Stelle benutzt werden:

```
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Info (#PCDATA)>

<!ELEMENT Koordinaten (X, Y)>
<!ELEMENT X (#PCDATA)> <!-- Echter Typ: unsigned int -->
<!ELEMENT Y (#PCDATA)> <!-- Echter Typ: unsigned int -->
```

3.4 Eine Wegenetzdatei als Beispiel

Diese Datei enthält ein (beinahe) minimales Beispiel-Wegenetz mit folgenden Informationen:

1. Eine Kartendatei,
2. zwei Knoten,
3. ein in beide Richtungen befahrbarer Straßenabschnitt zwischen diesen Knoten, dessen Definition einen zusätzlichen Stützpunkt benötigt, da die Straße einen Haken schlägt,
4. eine Klasse *Restaurant* und
5. eine Pizzeria, die an dem Straßenabschnitt aus (3.) liegt.

```
<WEGENETZ>

  <KARTE>hagen.jpg</KARTE>

  <KNOTEN>
    <Knoten KnotenId="knoten1">
      <Koordinaten>
        <X>100</X>
        <Y>200</Y>
      </Koordinaten>
    </Knoten>
    <Knoten KnotenId="knoten2">
      <Koordinaten>
        <X>500</X>
        <Y>600</Y>
      </Koordinaten>
    </Knoten>
  </KNOTEN>

  <KANTEN>
    <Kante KantenId="kantela" Typ="Strasse">
      <Start KnotenId="knoten1" />
      <Ziel KnotenId="knoten2" />
      <Stuetzpunkt>
        <X>200</X>
        <Y>400</Y>
      </Stuetzpunkt>
      <Name>Bahnhofstrasse</Name>
    </Kante>
```

```
<Kante KantenId="kantelb" Typ="Strasse">
  <Start KnotenId="knoten2" />
  <Ziel KnotenId="knoten1" />
  <Stuetzpunkt>
    <X>200</X>
    <Y>400</Y>
  </Stuetzpunkt>
  <Name>Bahnhofstrasse</Name>
</Kante>
</KANTEN>

<KLASSEN>
  <Klasse KlassenId="klassel">
    <Name>Restaurant</Name>
    <Attribut Typ="enum">
      <Name>Nationalitaet</Name>
      <Konstante>arabisch</Konstante>
      <Konstante>chinesisch</Konstante>
      <Konstante>deutsch</Konstante>
      <Konstante>franzoesisch</Konstante>
      <Konstante>griechisch</Konstante>
      <Konstante>italienisch</Konstante>
      <Konstante>japanisch</Konstante>
      <Konstante>mexikanisch</Konstante>
      <Konstante>spanisch</Konstante>
      <Konstante>thailaendisch</Konstante>
      <Konstante>tuerkisch</Konstante>
    </Attribut>
    <Attribut Typ="enum">
      <Name>Preisklasse</Name>
      <Konstante>Imbiss</Konstante>
      <Konstante>billig</Konstante>
      <Konstante>mittel</Konstante>
      <Konstante>teuer</Konstante>
    </Attribut>
  </Klasse>
</KLASSEN>

<PUNKTE>
  <Punkt KlassenId="klassel">
    <Koordinaten>
      <X>400</X>
      <Y>520</Y>
    </Koordinaten>
    <Anschluss>
      <KantenRef KantenId="kantela" />
      <KantenPos>0.75</KantenPos>
    </Anschluss>
    <Anschluss>
      <KantenRef KantenId="kantelb" />
      <KantenPos>0.25</KantenPos>
    </Anschluss>
    <Attributwert>italienisch</Attributwert>
    <Attributwert>mittel</Attributwert>
    <Name>Pizzeria Vesuvio</Name>
    <Info>Bahnhofsnaehe, kinderfreundlich</Info>
  </Punkt>
</PUNKTE>

</WEGENETZ>
```

4 Der mobile digitale Stadtführer

Der mobile digitale Stadtführer simuliert einen Palmtop mit integriertem GPS. Seine wesentlichen Funktionen bestehen darin, dem Benutzer einen Kartenausschnitt mit dem aktuellen Standort anzuzeigen, Recherchen nach interessanten Punkten in der Nähe zu ermöglichen und Routenplanung durchzuführen. Die dazu benötigten Informationen stehen in einer Wegenetzdatei (siehe Abschnitt 3) und einer dazu passenden Graphikdatei mit der Hintergrundkarte. Beide werden zu Beginn der Verwendung des MDS geladen. Man könnte sich vorstellen, daß es in Zukunft beispielsweise in Bahnhöfen Ladestationen gibt, an denen solche Daten auf einen Palmtop transferiert werden können.

Aufgabe der Routenplanung ist es, den *kürzesten* Weg zwischen zwei Positionen im Wegenetz zu ermitteln. Die dazu verwendeten Kosten, die einer Kante zugeordnet sind, werden zunächst durch die Länge des zugehörigen Linienzuges definiert. Allerdings ist es nicht sinnvoll, dasselbe Maß für alle Arten von Kanten zu verwenden, da man beispielsweise mit der U-Bahn meist schneller unterwegs ist als zu Fuß. Deshalb soll bei der Routenplanung die Entfernung noch je nach Art des Weges unterschiedlich gewichtet werden, wie in folgender Tabelle dargestellt:

Art des Weges	Gewichtung
Fußweg	1,0
Straße	0,3
Autobahn	0,1
Buslinie	0,4
Bahnlinie	0,25

Die Palmtop-Anwendung gibt dem Benutzer die Möglichkeit, die bei der Routenplanung zu berücksichtigenden Wegtypen zu wählen (siehe Abschnitt 5.11).

Der Kartenausschnitt mit dem aktuellen Standort kann in zwei Modi angezeigt werden. Zum einen gibt es die sogenannte *Zentralansicht*, die standardmäßig voreingestellt ist und die die Position des Benutzers immer in der Kartenmitte beläßt, so daß sich im Prinzip bei einer Bewegung des Benutzers die Karte bewegt. Zum anderen gibt es die sogenannte *nachgeführte Ansicht*, die den zurückgelegten Weg des Benutzers verfolgt und erst bei Erreichen eines vorgegebenen Abstandes zum Kartenrand auf einen anderen Kartenausschnitt übergeht.

Die Möglichkeiten zur visuellen Ausgabe auf einem Palmtop sind zur Zeit noch relativ beschränkt. So verfügt ein Original-Palm über ein Schwarzweiß-Display mit 160×160 Punkten. Wir wollen jedoch im Rahmen des Praktikums optimistisch annehmen, daß demnächst Geräte mit Farb-Displays mit 320×320 Punkten zur Verfügung stehen, so daß ein entsprechend großes Fenster auf dem Simulations-PC zur Ausgabe genutzt wird. Alle visuellen Ausgaben finden ausschließlich in diesem Fenster statt; dies betrifft nicht nur die Karte, sondern auch Informationen über interessante Punkte usw. Dabei befindet sich die Anwendung stets entweder im *Graphikmodus* oder im *Textmodus*. Die Umschaltung zwischen diesen Modi erfolgt automatisch in Abhängigkeit von den darzustellenden Informationen.

Eine weitere, noch optimistischere Annahme über den Palmtop der Zukunft besteht darin, daß er zuverlässig Spracheingaben verarbeitet. So wird es dem Benutzer des MDS ermöglicht, gesprochene Anfragen zu stellen. Dies wollen wir auf dem PC mit Hilfe eines Eingabefensters simulieren, das zusätzlich zum Hauptfenster auf dem PC-Desktop liegt und in das auf herkömmliche Weise Anfragen per Tastatur eingegeben werden können. Die möglichen Anfragen werden in Abschnitt 5 beschrieben. Analog zur *Spracheingabe* verfügt unser Wunschgerät auch über eine Komponente zur *Sprachausgabe*. Sie wird im Praktikum durch einfache Textausgabe simuliert.

Eine wesentliche Fähigkeit des „echten“ MDS liegt darin, ständig mittels GPS die eigene Position zu bestimmen. Im Software-Praktikum muß auch dies simuliert werden. Um der Realität einigermaßen nahe zu kommen, soll die aktuelle Position in der Simulation interaktiv bestimmbar sein, beispielsweise mit Hilfe der Cursortasten und/oder der Maus.

Zusätzlich zu den statischen Daten aus der Wegenetzdatei und der dynamischen Information über den eigenen Aufenthaltsort werden noch weitere dynamische Daten verarbeitet. Unser MDS kann sich in regelmäßigen Zeitabständen drahtlos mit einem *Personeninformationsserver (PIS)* in Verbindung setzen, um Namen und Aufenthaltsort weiterer mit dem System verbundener Personen zu erfahren. Diese können dann in der Karte angezeigt werden, und man kann in Anfragen Bezug auf sie nehmen (siehe Abschnitt 5).

Im Software-Praktikum muß auch der PIS als eigene Anwendung simuliert werden. Dies kann mit recht geringem Aufwand geschehen. Es reicht aus, den Benutzer in die Lage zu versetzen, Personen anzulegen und zu löschen und einige wenige Parameter zu jeder Person zu setzen, wie beispielsweise die Startposition und die Durchschnittsgeschwindigkeit. Anhand dieser Parameter kann der PIS dann zufällig eine Route durch das Wegenetz bestimmen.

Die drahtlose Verbindung des MDS mit dem PIS kann im Software-Praktikum mit Hilfe einer einfachen ASCII-Datei simuliert werden, die zu jeder gemeldeten Person deren Namen und ihre Position enthält. Diese Datei wird vom PIS in regelmäßigen Abständen, z.B. alle 5 Sekunden, aktualisiert und vom MDS ebenso regelmäßig ausgewertet.

5 Anfrageoperationen

Im folgenden beschreiben wir in Abschnitt 5.1 zunächst die Daten, auf denen Anfragen operieren. In den darauf folgenden Abschnitten werden die einzelnen Komponenten der Anfragesprache erläutert. Hervorzuheben ist nochmals, daß wir uns eine Spracheingabe zur Formulierung von Anfragen mittels einer einfachen Kommandosprache vorstellen sowie eine je nach Situation sprachliche, graphische oder textuelle Ausgabe zur Bekanntgabe von Anfrageergebnissen. Sprachliche Ein- und Ausgaben sollen allerdings textuell simuliert werden.

5.1 Relevante Daten für die Anfragekomponente

Für das Stellen von Anfragen sind folgende Daten von Bedeutung:

- unbenannte statische Objekte
- benannte statische Objekte

- benannte dynamische Objekte
- Kantennamen

Wir gehen davon aus, daß die Anwendung eine Menge von Objektklassen verwaltet. Jede Objektklasse enthält eine Menge von *unbenannten statischen Objekten*, deren *punktförmige* Position bekannt ist und deren Eigenschaften durch klassenspezifische Attribute der Datentypen *int*, *real*, *bool*, *string* und *enum* beschrieben werden. Es wird vorausgesetzt, daß jede Objektklasse standardmäßig ein Attribut *Name* vom Typ *string* besitzt, das das Objekt eindeutig charakterisiert (z.B. FernUniversität, Jakobikirche). Auf dieses Attribut kann mit Hilfe der Anfragesprache aber nur indirekt zugegriffen werden (siehe Abschnitt 5.9), indem nach Namen gesucht wird.

Ausgewählten Objekten einer Objektklasse kann man mit Hilfe eines Datendefinitionskommandos (siehe Abschnitt 5.1) einen expliziten Namen geben. Diese werden dann *benannte statische Objekte* genannt.

Benannte dynamische Objekte können nicht mit Hilfe eines Datendefinitionskommandos definiert werden, sondern werden von dem im Abschnitt 4 beschriebenen PIS mittels Angabe von Objekt-namen geliefert. So können wir uns zum Beispiel vorstellen, daß die aktuelle Position sowie die Bewegung einer uns interessierenden Person *Karl* im Graphikfenster angezeigt wird.

Die Kanten des Netzwerks können ebenfalls mit einem Attribut *Name* vom Typ *string* ausgestattet sein. Beispiele sind Straßennamen, Namen von S-Bahnlinien usw. Diese Namen dienen zum einen der Darstellung auf der Karte, sollen aber zum anderen auch zum Suchen mittels der Anfragesprache dienen (siehe Abschnitt 5.9). Es ist dann auch möglich, sich z.B. die *Feithstraße* anzeigen zu lassen.

5.2 Datendefinition

Betrachten wir nun den Datendefinitionsteil der Kommandosprache. *Teilklassen* beschreiben Objektklassen mit einschränkenden Eigenschaften. Interessieren wir uns beispielsweise für alle Pizzerien mit niedrigem Preisniveau, so kann man formulieren:

TEILKLASSE BilligPizzeria VON Pizzeria MIT Preiskategorie = preiswert

Die Klasse *BilligPizzeria* kann dann später wie eine Objektklasse in Anfragen verwendet werden. Sie stellt sozusagen eine eingeschränkte Sicht auf die Objektklasse *Pizzeria* dar. Hinter dem Schlüsselwort MIT sind ausschließlich boolsche Ausdrücke erlaubt, die wie üblich aus den logischen Operatoren UND, ODER und NICHT und *elementaren* boolschen Ausdrücken zusammengesetzt sind. Es gilt die Vorrangregel, daß NICHT höchste Priorität hat, dann UND und schließlich ODER. Dies wiederum bedeutet, daß Klammern benötigt werden und erlaubt sind. Statt der gesprochenen Formulierungen „Klammer auf“ und „Klammer zu“ sind der Übersichtlichkeit halber in der textuellen Simulation die bekannten Notationen „(“ und „)“ möglich. Elementare boolsche Ausdrücke haben die Form *b comp c* oder NICHT (*b comp c*), wobei *b* ein Attributname der in der Definition verwendeten Objektklasse, *c* ein Wert des zugehörigen Attributtyps (siehe Abschnitt 5.1) und $comp \in \{=, <, \leq, >, \geq\}$ sein muß.

Weiterhin ist es möglich, *benannte statische Objekte* zu definieren. Ein Beispiel ist die Definition

```
OBJEKT Reinoldikirche IST Kirche MIT Name = Reinoldikirche
```

Hinter dem Schlüsselwort OBJEKT ist ein Name anzugeben, der bisher noch nicht als Objektname ausgewählt wurde. Hinter dem Schlüsselwort MIT ist nur ein boolescher Ausdruck der Form $b = c$ erlaubt, wobei b ein Attributname der in der Definition verwendeten Objektklasse und c ein in der Datenbank existierender Wert des zugehörigen Attributtyps sein muß, der das Objekt eindeutig identifiziert.

5.3 Anfragearten

Bezüglich des Anfrageteils unserer Kommandosprache werden drei Arten von Anfragen unterschieden:

- *Statische Anfragen.* Diese Art von Anfrage wird *einmalig* ausgewertet. Auch bei Veränderung der eigenen Position sowie bei Veränderung der Position eines benannten dynamischen Objektes ändert sich das Anfrageergebnis nicht. Es besteht die Möglichkeit, sich alle Objekte des Anfrageergebnisses nacheinander anzuschauen. Das Stellen mehrerer solcher Anfragen ist erlaubt. Diese Anfrageart wird durch das Schlüsselwort ZEIGE eingeleitet.
- *Permanente Anfragen.* Diese Art von Anfrage wird *ständig* (bis auf Widerruf) ausgewertet. Ändert der Benutzer seine aktuelle Position, so wird die Anfrage automatisch neu berechnet und das Anfrageergebnis aktualisiert. Auch hier besteht die Möglichkeit, sich alle Objekte des Anfrageergebnisses anzuschauen. Das Stellen mehrerer solcher Anfragen ist erlaubt. Diese Anfrageart wird durch das Schlüsselwort BEOBACHTE eingeleitet.
- *Getriggerte Anfragen.* Häufig ist man daran interessiert, sich benachrichtigen zu lassen, sobald interessierende Objekte in die Nähe kommen. Auf diese wird dann durch ein akustisches Signal und z.B. durch eine blinkende Darstellung im Graphikfenster aufmerksam gemacht. Es handelt sich also auch hier um eine permanente Anfrage, die (bis auf Widerruf) ständig ausgewertet wird und deren Ergebnis stets aktualisiert wird. Das Stellen mehrerer solcher Anfragen ist erlaubt. Diese Anfrageart wird durch das Schlüsselwort MELDE eingeleitet.

Im folgenden werden durch Beispiele die oben genannten sowie weitere Kommandos eingeführt und erläutert.

5.4 Statische Anfragen

Mit dem Kommando

```
ZEIGE KLASSEN
```

wird im Textmodus eine scrollbare Liste (siehe Abschnitt 5.12) aller Namen der im System vorhandenen Objektklassen mit zugehörigen Attributen und deren Typen dargestellt.

Ist man daran interessiert, das von der eigenen, aktuellen Position nächstgelegene Objekt einer bestimmten Objektklasse (z.B. Kirchen) zu finden, so geschieht dies mit dem Kommando

ZEIGE Kirche

Gibt man den Namen n einer Objektklasse und eine Zahl m an, werden die entfernungs­mäßig m nächsten Objekte der Objektklasse n angezeigt. Wir können beispielsweise schreiben:

ZEIGE Kirche 3

Desweiteren ist es möglich, eine Aussage über die Art des Distanzmaßes zu machen, das benutzt werden soll. So wird beim Kommando

ZEIGE Kirche 3 RADIUS

die Euklidische Distanz gewählt, während beim Kommando

ZEIGE Kirche 3 PFAD

die Länge des kürzesten Pfades berücksichtigt wird. Es ist auch möglich, das Distanzmaß mittels

DISTANZ RADIUS oder DISTANZ PFAD

global einzustellen. Standardmäßig ist RADIUS voreingestellt. Ebenfalls ist die Anzeige der aktuellen Position eines benannten statischen oder dynamischen Objektes möglich. Beispiele hierfür sind

ZEIGE Reinoldikirche

ZEIGE Karl

Zu beachten ist, daß bei letztgenanntem Kommando nur die aktuelle Position von *Karl* angezeigt wird, die nicht aktualisiert wird. Eine Zahl als Ergänzung beider Kommandos macht keinen Sinn und ist daher nicht erlaubt.

Während wir bisher Objekte aus Objektklassen und benannte Objekte nur als Punkte darstellen lassen können, soll es auch möglich sein, zusätzlich die kürzesten Wege zu diesen Objekten angezeigt zu bekommen. Dies geschieht mit dem Kommando ZEIGE WEG. Beispiele sind

ZEIGE WEG Kirche

ZEIGE WEG Kirche 3

ZEIGE WEG Reinoldikirche

ZEIGE WEG Karl

Das zweite Beispiel berechnet die kürzesten Wege zu den drei nächstgelegenen Kirchen. Im letztgenannten Beispiel wird nur der augenblickliche Weg zu *Karl* angezeigt, der aber danach nicht aktualisiert wird.

Weiterhin ist es möglich, sich alle Objekte einer Objektklasse innerhalb eines vorgegebenen Abstands anzeigen zu lassen. Beispiele sind

ZEIGE Pizzeria BIS 500 METER

ZEIGE Kirche BIS 2 KILOMETER

Neben der positionsmäßigen Anzeige eines oder mehrerer Objekte soll es allgemein auch möglich sein, Objekte von Anfrageergebnissen nacheinander auszuwählen und Zusatzinformationen zu erhalten. Mit Hilfe des Kommandos

WÄHLE Pizzeria

wird zum Beispiel aus der Menge der gerade angezeigten Pizzerien die nächstgelegene Pizzeria ausgewählt und durch Hervorhebung angezeigt. Mit dem Kommando

NÄCHSTE Pizzeria

kann man jeweils durch die nach aufsteigendem Abstand geordnete Liste dieser Pizzerien laufen und eine weitere Pizzeria auswählen. Neben dem Schlüsselwort NÄCHSTE sind auch die Schlüsselworte NÄCHSTER und NÄCHSTES erlaubt (z.B. NÄCHSTER Bus, NÄCHSTES Autohaus).

Die Attribute eines gerade ausgewählten Objekts können mittels des Kommandos

DATEN

erfragt werden. Dann wird das Graphikfenster durch ein reines Textfenster ersetzt, das die Informationen über das Objekt anzeigt. Eine Umschaltung auf das Graphikfenster und die Kartendarstellung erfolgt mit dem Kommando

GRAPHIK

Außerdem gibt es noch einige statische Kommandos, mit denen es möglich ist, die Karte sowie das zugrundeliegende Netzwerk explizit einzublenden oder auszublenden. Dies geschieht mit den vier Kommandos

ZEIGE KARTE

ZEIGE NETZ

VERBIRG KARTE

VERBIRG NETZ

5.5 Permanente Anfragen

Permanente Anfragen werden mit dem Kommando BEOBACHTE eingeleitet. Grundsätzlich sind wie beim ZEIGE-Kommando Objektklassen, benannte statische Objekte und benannte dynamische Objekte als Argumente erlaubt. Wir dürfen also beispielsweise formulieren:

BEOBACHTE Kirche

BEOBACHTE Kirche 3

BEOBACHTE Kirche 3 RADIUS

BEOBACHTE Kirche 3 PFAD

BEOBACHTE Pizzeria BIS 500 METER

BEOBACHTE Kirche BIS 2 KILOMETER

Die Idee ist, daß die Anfrage permanent in Abhängigkeit von der Position des Benutzers und dessen Bewegungen ausgewertet und das Anfrageergebnis stets aktualisiert wird. Es werden also ständig das entfernungsmaßig nächstgelegene Objekt bzw. die nächsten m Objekte einer Objektklasse bzw. die sich innerhalb einer bestimmten Distanz befindlichen Objekte ermittelt und angezeigt.

Für benannte statische Objekte gilt, daß die Bedeutung eines Kommandos wie zum Beispiel

BEOBACHTE Reinoldikirche

die gleiche ist wie die eines Kommandos „ZEIGE Reinoldikirche“. Obwohl keine dynamischen Bewegungen im Spiel sind, wird dieses Kommando aus Gründen der Einfachheit und Bequemlichkeit erlaubt.

Bei benannten dynamischen Objekten wird ständig bzw. in vertretbaren Zeitabständen die Position eines solchen Objekts im Graphikfenster aktualisiert. Beispielsweise können wir formulieren:

BEOBACHTE Karl

Neben der permanenten Anzeige und Aktualisierung der Position von Objekten kann auch ständig der kürzeste Weg zu einem Objekt in Abhängigkeit von der eigenen Position und/oder der Position des interessierenden Objekts bzw. der interessierenden Objekte angezeigt werden.

BEOBACHTE WEG Kirche

BEOBACHTE WEG Kirche 3

BEOBACHTE WEG Reinoldikirche

BEOBACHTE WEG Karl

Die Aktualisierung der Darstellung im Graphikfenster ist bei Objekten einer Objektklasse sowie bei benannten statischen Objekten nur von der Bewegung des Benutzers abhängig. Bei benannten dynamischen Objekten (wie z.B. *Karl*) muß auch deren Bewegung in Betracht gezogen werden.

Bei allen permanenten Anfragen kann der WÄHLE-NÄCHSTE-Mechanismus zum Durchlaufen des aktuellen und aktualisierten Anfrageergebnisses angewendet werden. Auch die globale Umschaltung des Distanzmaßes ist möglich.

5.6 Getriggerte Anfragen

Getriggerte Anfragen sind permanente Anfragen, die ihre Anfrageergebnisse dem Benutzer zusätzlich durch ein akustisches Signal und eine blinkende Darstellung im Graphikfenster anzeigen. Sie werden durch das Kommando MELDE eingeleitet.

Möchte man sich beispielsweise benachrichtigen lassen, sobald ein oder mehrere interessierende Objekte in die Nähe des Benutzers kommen, so kann man dies mit einer getriggerten Anfrage formulieren. Auf diese Objekte wird dann mit einem akustischen Signal und z.B. mit einer blinkenden Darstellung im Graphikfenster reagiert. Beispiele sind:

MELDE Museum BIS 500 METER

MELDE U-Bahn-Station BIS 200 METER

Um die eingehenden Meldungen nacheinander betrachten zu können, kann man mit dem Kommando

MELDUNGEN

in den Textmodus umschalten und die erste Meldung auswählen.

Mit dem Kommando

NÄCHSTE MELDUNG

wird die nächste Meldung ausgewählt. Die Darstellung einer Meldung beinhaltet den Namen der Objektklasse des gemeldeten Objekts sowie alle seine Attribute.

5.7 Zurücknehmen von Anfragen

Alle Anfragearten fügen Anfrageergebnisse zu den bereits graphisch dargestellten Anfrageergebnissen hinzu. Weil hierdurch das Graphikfenster mit Informationen „überladen“ werden kann und weil zudem der Benutzer das Interesse an bestimmten Anfrageergebnissen verlieren kann, muß es möglich sein, Anfrageergebnisse auch zu löschen. Hierzu dient das (bereits oben erwähnte) Kommando VERBIRG, das es (auch) erlaubt, die Darstellung von Objekten einer bestimmten Objektklasse sowie von benannten statischen und dynamischen Objekten zu löschen. Beispielsweise kann man formulieren:

VERBIRG ALLES
VERBIRG KLASSEN
VERBIRG Kirche
VERBIRG Reinoldikirche
VERBIRG Karl

Das erste Kommando bewirkt hierbei ein Löschen aller Anfrageergebnisse. Das zweite Kommando löscht alle Anfrageergebnisse, die durch Nennen eines Objektklassennamens ermittelt wurden. Zu beachten ist, daß mit dem Verbergen von Anfrageergebnissen auch die zugehörige Anfrage ungültig wird.

Meldungen bleiben solange aktuell, bis sie vom Benutzer gelöscht werden. Dies geschieht einzeln mit dem Kommando

VERBIRG MELDUNG

im MELDUNGEN-Textmodus oder aber global mit dem Kommando

VERBIRG MELDUNGEN

5.8 Nacheinanderausführung von Anfragen

Prinzipiell ist es möglich, nacheinander mehrere Anfragen bezüglich der gleichen Objektklasse oder des gleichen benannten statischen oder dynamischen Objekts zu stellen. Grundsätzlich soll dann das zuletzt genannte Kommando gelten. Die Darstellung früherer Anfrageergebnisse bezüglich der gleichen Objektklasse bzw. des gleichen Objekts ist dann zu löschen. Im Falle der nacheinander formulierten Kommandos

ZEIGE Kirche 3
BEOBACHTE Kirche
MELDE Kirche BIS 200 METER

wird zunächst das Ergebnis der statischen Anfrage dargestellt und gelöscht und dann das Ergebnis der dynamischen Anfrage dargestellt. Später wird auch dieses Anfrageergebnis gelöscht und durch das Anfrageergebnis des MELDE-Kommandos ersetzt.

5.9 Das FINDE-Kommando

Alle bisherigen Kommandos beziehen sich auf punktförmige Objekte und haben keinen direkten Bezug zum Wegenetz. Um auch von der Anfrageseite her zumindest in geringem Maße die Informationen des Wegenetzes ausnutzen zu können und um auch unbenannte statische Objekte von Objektklassen einfacher und direkter finden zu können, wird das FINDE-Kommando eingeführt.

Häufig fehlt einem Benutzer die Kenntnis der Lage von Straßen. Dann bewirkt zum Beispiel das Kommando

FINDE Feithstraße

daß im Wegenetz und in allen Objektklassen nach Kanten bzw. nach unbenannten Objekten gesucht wird, deren Attribut *Name* den Wert *Feithstraße* hat. Diese Straße wird dann mit allen Knoten und Kanten graphisch hervorgehoben. Das Kommando

FINDE WEG Feithstraße

zeigt den kürzesten Weg von der aktuellen Position des Benutzers zu einem Knoten der Feithstraße an.

Hinter FINDE bzw. FINDE WEG steht also grundsätzlich ein String, der auch Leerzeichen enthalten darf. So können wir beispielsweise formulieren:

FINDE Berliner Allee

5.10 Das FÜHRE-Kommando

Nach den Kommandos ZEIGE WEG oder FINDE WEG soll es möglich sein, sich mit Hilfe des Systems zum Ziel führen zu lassen. Die geschieht mit Hilfe des Kommandos

FÜHRE

Mittels Sprachausgabe, die textuell simuliert wird, erhält der Benutzer Richtungsangaben gemäß dem zuvor berechneten kürzesten Weg. Kommandos der Sprachausgabe sind zum Beispiel

GERADEAUS
NACH LINKS
NACH RECHTS
ZURÜCK

und weitere geeignete Richtungsangaben. Geeignete Fehlermeldungen und Korrekturangaben müssen ebenfalls ausgegeben werden können.

5.11 Auswahl von Teilnetzen

Das der Anwendung zugrundeliegende Wegenetz ist in fünf Teilnetze unterteilt, nämlich für Fußwege, Straßen, Autobahnen, Buslinien und Bahnlinien. Standardmäßig werden bei der Berechnung kürzester Wege die Teilnetze für Fußwege, Buslinien und Bahnlinien benutzt. Um dies zu ändern, können mit Hilfe des Kommandos BENUTZE auch andere Kombinationen von Teilnetzen ausgewählt werden. Beispiele hierfür sind

BENUTZE Autobahn, Straße
BENUTZE Fußweg, Buslinie

5.12 Scrollmechanismus für Textfenster

Da es unter Umständen ein Problem sein kann, im Textmodus Daten in ihrer Gesamtheit im Textfenster darzustellen, wird ein Scrollmechanismus angeboten. Mittels des Kommandos

WEITER

wird weitergeblättert und mittels des Kommandos

ZURÜCK

zurückgeblättert.

5.13 Ansichtsarten sowie Vergrößerungs- und Verkleinerungsoperationen

Wie in Abschnitt 4 bereits beschrieben, existieren die beiden Ansichtsarten *Zentralansicht* (Position des Benutzers immer in der Kartenmitte) und *nachgeführte Ansicht* (zurückgelegter Weg bleibt sichtbar). Mit dem Kommando

ANSICHTSWECHSEL

kann man zwischen den beiden Ansichtsarten umschalten.

Desweiteren kann man mit dem Kommando

ZOOM IN

in die Karte hineinzoomen und einen kleineren Ausschnitt um den Benutzer herum vergrößert darstellen. Umgekehrt erlaubt das Kommando

ZOOM OUT

einen größeren Ausschnitt um den Benutzer herum verkleinert darstellen. Mit dem Kommando

ZOOM STANDARD

gelangt man zur Standardeinstellung zurück.

6 Anforderungsdefinitions- und Entwurfsrichtlinien

Der Entwurf ist gemäß den Ihnen bekannten Methoden des Kurses *Software Engineering I* zu erstellen. Ihre Aufgabe ist es zunächst, eine Anforderungsdefinition und einen Entwurf für ein System „Mobiler Digitaler Stadtführer“ zu erstellen. Das Ergebnis muß folgende Teile enthalten:

- eine formale Analyse der Aufgabenstellung mit den im Kurs *Software Engineering I* vorgestellten Hilfsmitteln: ER-Diagramm, Datenflußmodellierung, Einteilung in Teilprozesse und Spezifikation der Prozesse,
- eine grobe Aufteilung des Gesamtsystems in Teilsysteme und Module sowie die graphische Darstellung der Abhängigkeiten (Benutzt-Beziehungen) zwischen den Teilsystemen und Modulen,
- eine Spezifikation der Module durch die Beschreibung der Modulschnittstellen und der Modulsemantik, aus der Art und Funktion der Module und der in den Schnittstellen übergebenen Objekte (Prozeduren, Variablen, Typen, ...) klar hervorgehen müssen,
- einen Entwurf der Benutzeroberfläche mit Beschreibung des Layouts, der Komponenten und der möglichen Interaktionen,
- eine Liste von Fragen, die mit dem „Auftraggeber“ noch zu klären sind.

Bitte beachten Sie, daß Sie bei Ihrem Entwurf keine Rumpfspezifikationen erstellen müssen.

Der nächste Schritt des Entwurfs gemäß dem Kurs *Software Engineering I* wäre das Erstellen von Prozedurköpfen für Modula-2. Da das Software-Praktikum in *Java* durchgeführt wird, die Methoden des objektorientierten Entwurfs aber erst im Kurs *Software Engineering II* behandelt werden, können Sie auf das Erstellen dieser Prozedurköpfe verzichten. In der ersten Präsenzphase werden wir gemeinsam die Vorgehensweise besprechen, wie ein Übergang von diesem Entwurf zur Implementierung in *Java* geschaffen werden kann.

7 Programmierrichtlinien und Dokumentation

Es werden keine Programmierrichtlinien an die Hand gegeben. Allerdings sollten innerhalb jeder Gruppe solche Richtlinien existieren, damit jeder die Programme seiner Gruppe lesen und verstehen kann. Daher muß jede Gruppe eigene Programmierrichtlinien erstellen und schriftlich fixieren.

Ihnen wird für den Zeitraum des Praktikums von der FernUniversität ein Laptop mit einer Installation des Java SDK 1.3.1 zur Verfügung gestellt. Darüber hinausgehende Tools zur Software-Entwicklung sind nicht installiert. Die Installation und Verwendung von frei verfügbaren Entwicklungswerkzeugen ist Ihnen jedoch freigestellt.

Die Dokumentation des Programms ist mit Hilfe von *Javadoc* (im SDK enthalten) zu erstellen. Da mit *Javadoc* lediglich die Schnittstellen dokumentiert werden, ist der Programmcode zusätzlich hinreichend mit Kommentaren zu versehen.

Schließlich ist ein Benutzerhandbuch zu erstellen, das die Handhabung Ihres Programms ausführlich erläutert.